Attached is a clean copy of the Substitute Specification filed on December 3, 3002, in accordance with 37 C.F.R. §§ 1.121(b) (3) and 1.125(c). The substitute specification contains no new matter.

SUBSTITUTE SPECIFICATION

TITLE OF THE INVENTION

Round Key Generation for AES Rijndael Block Cipher

CROSS-REFERENCE TO RELATED APPLICATIONS

None

BACKGROUND OF THE INVENTION

[0001] The present invention relates to methods and apparatus for implementation of the Advanced Encryption Standard (AES) algorithm and in particular to methods and apparatus for real-time generation of the round keys required during the encryption and decryption rounds of the algorithm.

[0002] The invention has particular, though not exclusive, application in cryptographic devices, such as those installed in smart cards and other devices where processor and memory resources are limited.

[0003] The AES (Rijndael) algorithm may be implemented using a 128-bit, a 192-bit, or a 256-bit key operating on successive 128-bit blocks of input data.

During implementation of an encryption operation or a decryption operation according to the AES algorithm (hereinafter, generally a "cryptographic operation"), the original or "initial" key must be expanded to provide a round key for each successive round of the encryption or decryption operation. The number of rounds (N_r) is 10 for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys.

[0004] Thus, the expanded round key is the size of the initial key multiplied by (N_{r+1}). In the case of a 128-bit key, the expanded key comprises 128 × 11=1408 bits; for the 192-bit key, the expanded key comprises 128 × 13=1664 bits; and for the 256-bit key, the expanded key comprises 128 × 15=1920 bits.

[0005] Storage of this expanded key consumes a significant amount of memory space in cryptographic engines, which is particularly significant in certain applications, such as the provision of cryptographic engines on smartcards and the like where memory space is limited. Provision of this space is not strictly necessary if round keys can be generated during operation of the cryptographic engine without causing delay thereto.

BRIEF SUMMARY OF THE INVENTION

[0006] The present invention is directed towards a key expansion method and apparatus to implement the round key generation function in real time using a substantially reduced memory allocation than existing techniques.

[0007] The present invention recognizes that real time generation of the successive round keys can be performed in parallel with execution of the encryption or decryption algorithm in the cryptographic engine, have little impact on the execution time of the encryption or decryption process, and have reduced amounts of hardware.

[0008] According to one aspect, the present invention provides a method of generating successive round keys of an expanded key from an initial cryptographic key for use in an encryption and/or decryption engine, comprising the steps of:

[0009] storing the N_k words of the initial key in N_k locations of a memory;

[0010] providing the initial key to a cryptographic engine for performing a first cryptographic round;

[0011] repeatedly retrieving a selected first word and a selected second word of the expanded key, at least one of which is retrieved from the memory, and generating from the selected first and second words a successive subsequent word of the expanded key;

[0012] providing the generated words of the expanded key to the cryptographic engine as round keys for performing subsequent cryptographic rounds; and

[0013] storing successive ones of the generated subsequent words in the memory by cyclically overwriting previously generated words of the expanded key.

[0014] According to another aspect, the present invention provides a round key generator for generating successive round keys of an expanded key from an initial cryptographic key for use in an encryption and/or decryption engine, comprising:

[0015] a memory for storing the N_k words of the initial key;

[0016] an expansion processor for repeatedly retrieving a selected first word and a selected second word of the expanded key, at least one of which is retrieved from the memory, and generating from the selected first and second words a successive subsequent word of the expanded key;

[0017] means for providing the generated words of the expanded key to the cryptographic engine as round keys for performing subsequent cryptographic rounds; and

[0018] means for storing successive ones of the generated subsequent words in the memory by cyclically overwriting previously generated words of the expanded key.

[0019] According to another aspect, the present invention provides an AES round constant function generator comprising a shift register having:

[0020] a first control input for causing a left shift of the register contents;

[0021] a second control input for causing a right shift of the register contents; and

[0022] a third control input for causing a preset of the shift register contents to one of several possible values.

BRIEF DESCRIPTION OF THE DRAWINGS

[0023] Embodiments of the present invention will now be described by way of example and with reference to the accompanying drawings in which:

[0024] FIG. 1 is a flow diagram illustrating implementation of an encryption operation using the AES block cipher algorithm;

[0025] FIG. 2 is a flow chart of the AES round key schedule used to generate the expanded encryption key that provides the plural round keys required during an encryption operation;

[0026] FIG. 3 is a schematic block diagram of a round key generator according to the present invention;

[0027] FIG. 4 is a schematic block diagram of the key expansion processor for generating the succession of round keys during encryption; and

[0028] FIG. 5 is a schematic block diagram of the key expansion processor for generating the succession of round keys during decryption.

DETAILED DESCRIPTION OF THE INVENTION

[0029] The AES algorithm for encryption of plaintext to ciphertext is shown in FIG. 1. The AES algorithm may be implemented using a 128-bit, a 192-bit, or a 256-bit key operating on successive 128-bit blocks of input data. FIG. 1 will now be described in the context of the basic implementation using a 128-bit key.

[0030] An initial 128-bit block of input plaintext 10 is XOR-combined 11 with an original 128-bit key 12 in an initial round 15. The output 13 from this initial round 15 is then passed through a number of repeated transform stages, in an encryption round 28 which include the Sub Bytes transform 20, the Shift Rows transform 21, and the Mix Columns transform 22 in accordance with the defined AES algorithm.

[0031] The output from the Mix Columns transform 22 is XOR-combined 23 with a new 128-bit round key 26, which has been derived from the initial key 12. The output from this XOR-combination is fedback to pass through the encryption round 28 a number of further times.

[0032] For each successive iteration through the encryption round 28, a new round key 26 is derived from the existing round key 26 according to the AES round key schedule.

[0033] The number of iterations (N_{r-1}) of the encryption round 28 is 9 where a 128-bit encryption key is being used, 11 where a 192-bit encryption key is being used, and 13 where a 256-bit encryption key is being used.

[0034] After the requisite number (N_{r-1}) of encryption rounds 28, a final round, N_r, is entered under the control of decision box 24. The final round 30 comprises a further Sub Bytes transform 31, a further Shift Rows transform 32, and a subsequent XOR-combination 33 of the result with a final round key 36 generated 35 from the previous round key. The output therefrom comprises the cipher text output 39 of the encryption algorithm.

[0035] It will be noted from FIG. 1 that the implementation of the AES encryption algorithm requires generation of new round keys 25 from the initial key 12 ready for each round 28, 30.

[0036] Throughout the present specification, the keys will be expressed in terms of the number, N_k, of 32-bit words. For an initial 128-bit encryption key 12, ie. 4×32-bit words, $N_k=4$, and the "expanded" key comprises 11×4 32-bit words, or 44 words, written as W (0) . . . W (43). For an initial 192-bit encryption key ($N_k=6$), the expanded key rises to 13×4 32-bit words, or 52 words, written as W(0) . . . W(52). For an initial 256-bit encryption key ($N_k=8$), the expanded key rises to 15×4 32-bit words, or 60 words, written as W(0) . . . W(59).

[0037] During execution of the AES decryption algorithm, the round keys are the same as for encryption, but presented in the reverse order.

[0038] With reference to FIG. 2, the general AES key expansion algorithm for generating the successive round keys will now be described, in the context of a 128-bit key (number of words in the key, $N_k=4$). It will be understood that the technique also applies to 192-bit ($N_k=6$) and 256-bit ($N_k=8$) keys.

[0039] The initial key 50, comprising four 32-bit words W(0), W(1), W(2) and W(3) is loaded into suitable memory locations 51_0, 51_1, 51_2, 51_3. In a conventional implementation, the memory includes sufficient space, at 51_n to accommodate all words of the expanded key, once it is generated.

[0040] Each new sequence of four words in the expanded key comprises a new round key and will be referred to as a "stretch". More generally, a stretch is W(i) to W(i+N_k) where i is an integer multiple of N_k, minus 1 (0, 3, 7 etc for N_k=4; 0, 7, 15 for N_k=8). At the outset, the only stretch is the initial key 50, and the first task is to generate the first word of a new stretch, the decision box 53 thereby indicating path "yes".

[0041] In the initial pass of the key expansion algorithm, the last word of the preceding stretch (51_3) is extracted (at 52) , the bits are left shifted (step 54), and then transformed according to the AES key expansion algorithm using an S-box look-up 55. The S-box function is the same as that for the AES SubBytes transform 20 (FIG. 1). The resulting 32-bit output 56 is transformed by XOR-combination 57 of the first eight bits only with a round constant R_{con} 58 defined in the AES key schedule. The output 60 from this operation is then XOR-combined 62 with the first word of the preceding stretch (ie. 51_0) and this result--W(4)--written 63 to memory at 51_4.

[0042] In the second pass through the flow diagram, the next word W(5) of the second stretch is derived. This being the second word of a stretch, the left hand path of the flow diagram is taken, the newly generated word, W(4), at 51_4, being copied 61 directly to the Wtmp buffer 60 ready for simple XOR-combination 62 with the next word 51_1 of the initial key 50. The new generated word W(5) is written (at 63) to memory 51_5.

[0043] The procedure repeats the left hand path a further two times, generating the last two words W(6) and W(7) of the second stretch, before recommencing the cycle for the third stretch, using the right hand path.

[0044] In effect, it will be seen that each word of each new stretch is the XOR-combination of its immediately preceding word and the word in the corresponding position of the preceding stretch, with the exception of the first word in each stretch. For the first word in each stretch, it is a function of the immediately preceding word that is used, rather than the immediately preceding word itself, the function being executed according to steps 54-59 of FIG. 2.

[0045] The principle deployed for 192-bit (N_k=6) and 256-bit (N_k=8) keys is the same, except that each stretch is respectively six words or eight words in length.

[0046] Each successive group of four words is used as the round key for each successive round 28, 30 of the encryption procedure of FIG. 1. During decryption, the round keys are applied in reverse order.

[0047] In one aspect, the present invention recognizes that it is only necessary to retain in memory the N_k words of the original key together with the most recent N_k words of the expanded round key at any one time. The most recently generated four words (or, more generally, four successive words in the currently held N_k words) are fed into the encryption engine at steps 23 or 33, while the held N_k words are used to generate the new stretch as described in FIG. 2.

[0048] Providing that the new stretches are generated fast enough to keep up with the encryption engine, and maintained in synchronism therewith (within the tolerance of the difference of a stretch length (N_k=4, 6 or 8) and round key length (=4) so that the most recently generated stretch includes the round key which is currently required in the encryption engine, then very limited memory capacity and buffer requirements only need be provided.

[0049] With reference to FIG. 3, the round key generator 100 comprises a RAM sector 101 that is divided into equal parts 102, 103, each part having a size of, for example, 4×32 bit words (for the 128-bit key algorithm), 6×32 bit words (for the 192 bit key generator) or 8×32 bit words (for the 256 bit key algorithm). Throughout the following description, a round key generator 100 capable of handling a 256-bit key algorithm will be assumed, this being adaptable to accommodate processing of smaller key lengths.

[0050] For convenience, the two parts 102, 103 will be referred to as the lower half 103 and the upper half 102. The respective halves are referenced for read access by an OffSetHiRd pointer 105 via multiplexer 104. For OffSetHiRd=0, lower half 103 is read; for OffSetHiRd=1, upper half 102 is read. In the lower half 103 of the RAM 101, the initial encryption key 50 is stored in locations W_0 to W_7 (ie. the first stretch W (0) . . . W (7) for N_k=8); in the upper half 102, the new calculated stretch, e.g. W (8) . . . W (15) is stored in corresponding upper half locations W_0 . . . W_7. A pointer OffSetHiWr (not shown) may be used to point to the memory half being written to). As each successive stretch is generated and used in the encryption engine, the next stretch values (e.g. W (16) . . . W (23)) are calculated and overwritten into the upper half 102.

[0051] The individual locations W_0 . . . W_7 (lower half) or W_1 . . . W_7 (upper half) are referenced for read and write operations by an OffSetCnt counter 111 which is a three-bit counter that points to one of the word locations in the upper half and/or the corresponding location in the lower half. In general, the OffSetCnt counter 111 is implemented as a modulo Nk up/down counter.

[0052] A round key counter 110 maintains a count of the currently calculated round key (i.e. the current stretch). A state machine 106 maintains overall control of the round key generation process, and an expansion processor 107 performs the computation of the expanded round key values (words).

[0053] When the encryption operation for the current plaintext block is complete, the procedure may be recommenced from the encryption key in the lower half 103. Alternatively, if a decryption operation is required, the first round key of the decryption cycle comprises the most recently calculated round key from the upper RAM half 102, which may be moved into the lower half, or read from the upper half. Successive decryption round keys are calculated in similar manner. At the completion of the decryption round key generation operation, the original encryption key is returned and can be restored to or retained in the lower half of RAM 101 for a subsequent encryption operation.

[0054] FIG. 4 shows a block diagram of the expansion processor 107. The expansion processor 107 comprises a first 32-bit register W, shown at 120, and a second 32-bit register W_{tmp}, shown at 121. Each register W, W_{tmp} can be filled directly from the RAM 101. A 32-bit, two input multiplexer 122 also allows the filling of W_{tmp} via a feedback line 123. The expansion processor 107 further includes special processing logic 150 for effecting the transforms RotateWord 154, SubWord 155, R_{con} 158 as described in connection with transforms 54, 55, 58 in FIG. 2. XOR gate 157 is analogous to XOR gate 57 in FIG. 2. A 32-bit multiplexer 124 selects output from either the special processing logic 150 or direct from register W_{tmp} 121 to provide input to 32-bit wide XOR gate 162.

[0055] At the start of an encryption operation, the initial key 50 (W(0) . . . W(7)) is loaded into RAM 101 into the lower half 103, positions W_0 . . . W_7. The first word W(0) of the initial key 50 is loaded into the buffer 120 from RAM 101 and the last word $W(N_k\text{-}1)$ of the initial key 50 is loaded into buffer W_{tmp} 121. More generally, for successive rounds of encryption, W(i) is loaded into buffer 120, and the last calculated value of $W(i+N_k)$ is stored in W_{tmp} 121.

[0056] As defined with reference to FIG. 2, during a key expansion process for encryption, one the following equations applies to the generation of each new word W(i) of the expanded round key:

[0057] For all i except those below (i.e. no special processing 150),[0058] Rule 1: $W(i) = W(i-N_k) \oplus W(i-1)$

[0059] When i mod N_k=0 (the beginning of each stretch),

[0060] Rule 2: $W(i) = W(i-N_k) \oplus \text{SubWord}(\text{RotWord}(W(i-1))) \oplus R_{con}(i/N_k)$

[0061] When i mod N_k=4 and N_k=8 (the middle cycle of each 8 word stretch),

[0062] Rule 3: $W(i) = W(i-N_k) \oplus \text{SubWord}(W(i-1))$

[0063] where:

[0064] RotWord(W_{tmp}) is a bytewise rotation of W_{tmp},

[0065] SubWord is the AES S-box transform,

[0066] R_{con} is the round constant as defined in the AES standard, which is applied only to the first byte of the first word in each stretch, while the other bytes are passed unchanged,

[0067] i=0 . . . $4N_r$+3,

[0068] i.e.

[0069] i=0 . . . 43for N_k=4;

[0070] i=0 . . . 51 for N_k=6 and

[0071] i=0 . . . 59 for N_k=8.

[0072] In other words, for the first word of each new stretch, the special processing of steps 54-59 is applied and $W(N_k)$ is calculated as the XOR-combination 62 of W(0) from register 120 and the transformed $W(N_k\text{-}1)$. For the middle word of each stretch when N_k=8, the special processing only of step 55 is applied. For other words in each stretch, the contents of register

120 and register 121 are XOR-combined directly without the special processing of steps 54 to 59.

[0073] With reference to FIG. 4 and FIG. 5, register W is loaded with W(0) and register W_{tmp} is loaded with $W(N_k\text{-}1)$ [e.g. W(7) for Nk=8]. Then the result of the calculation, being $W(N_k)$, [e.g. W(8)], is output from XOR gate 162 and stored in both RAM 101 [e.g. at location W_0, upper half] and in register W_{tmp} 121. Then, register W is loaded with W(1), while register W_{tmp} holds $W(N_k)$, [e.g. W(8)]. Then W (N_k+1) [e.g. W (9)] is calculated and stored in RAM 101 [at location W_1, upper half] and in register W_{tmp}.

[0074] In general, register W is loaded from RAM 101 with W(i), while register W_{tmp} holds the value of $W(i+N_k\text{-}1)$. Then $W(i+N_k)$ is calculated and stored both in RAM 101, at position $W_{(i+Nk)}$ mod 8, upper half (i.e. new values are stored cyclically in the upper half 102), and in W_{tmp}.

[0075] The key expansion process runs in parallel with the encryption processor 130 which preferably works word-by-word rather than on blocks 128 bits wide. In this manner, the content of W can be passed directly to the encryption processor to be used immediately as input for the encryption process. In the alternative, the encryption processor 130 may be coupled directly to access RAM 101 to retrieve the required words of the round key. This configuration allows more flexibility in the relative timing of the cycles of operation of the encryption engine 130 and the expansion processor 107.

[0076] For each cycle of operation, the new value of W_{tmp} is such that:

[0077] $W_{tmp} = W_{tmp} \oplus W$, except for the following cases:

[0078] When i mod N_k=0,

[0079] then W_{tmp} = SubWord (RotWord (W_{tmp})) $\oplus$ R_{con} (i/N_k) $\oplus$ W

[0080] When i mod N_k=4 and N_k=8,

[0081] then W_{tmp} = SubWord (W_{tmp}) $\oplus$ W

[0082] During the key expansion process, the pointer OffSetHiRd 105 effectively points to a base word location in RAM 101 either in the upper half 102 and the lower half 103. Control of the read locations is implemented by this one-bit pointer which respectively selects the read half of the memory. Thus, during the first cycle of key expansion (during computation of the second stretch), the initial key words W (0) . . . W (7) are read from the lower half 102, i.e. the read flag 105 selects OffSetLo. During encryption key expansion, new values of the round keys are always written to the upper half 102.

[0083] At the start, the following initialization settings apply:

[0084] OffSetCnt=0, OffSetHiRd=0, OffSetHiWr=1, RndCnt=4N_r+3.

[0085] The RAM 101 is read at address W_{Nk-1}, determined by OffSetHiRd and OffSetCnt (i.e. OffSetCnt+Nk-1), and stored in W_{tmp}.

[0086] Then the following procedure is executed N_k times:

[0087] 1. Read the RAM at $W_{OffSetCnt}$ from the lower half, and store it in W.

[0088] 2. Generate the next expanded key word and write it to Wtmp and to the memory at $W_{OffSetCnt}$ in the upper half 102.

[0089] 3. Increment OffSetCnt and decrement RndCnt.

[0090] 4. Update R_{con} only after the first cycle of the N_k cycles.

[0091] All words of the initial key from the lower half 103 have now been used. OffSetHiRd is set to 1, so that all subsequent round key words are read from the upper half 102. For example, for N_k=8, the memory at address W_8 contains W (8).

[0092] Now, the following procedure is executed repeatedly until RndCnt = N_k-1.

[0093] 1. Read RAM at OffSetCnt from the upper half (OffSetHi=1) and store it in W.

[0094] 2. Generate the next Round Key word and write it to W_{tmp} and to the RAM at OffSetCnt in the upper half.

[0095] 3. Update R_{con} when OffSetCnt=0

[0096]4. Increment OffSetCnt and decrement RndCnt.

[0097] For N_k=4, the last calculation is W (43) = W (39) ⊕ W (42). OffSetCnt = 43 mod 4 = 3.

[0098] For Nk=6, the last calculation is W (51) = W (45) ⊕ W (50). OffSetCnt = 51 mod 6 = 3.

[0099] For N_k=8, the last calculation is W (59) = W (51) ⊕ W (58). OffSetCnt = 59 mod 8 = 3.

[0100] So, independent of Nk, the last Round Key word is always stored at OffSetCnt=3.

[0101] At this point, the last N_k round key words are used by the encryption processor 130, but there are no more Round Key words to be generated by the expansion processor. Thus, the following procedure is executed repeatedly until RndCnt=0:

[0102] 1. Read the RAM at $W_{OffSetCnt}$ from the upper half and store it in W.

[0103] 2. Increment OffSetCnt and decrement RndCnt.

[0104] It will be noted that the lower half 103 of the RAM 101 now contains the initial encryption key (N_k words), and the upper half 102 of RAM now contains the final N_k words of the expanded key. The final N_k words of the expanded key are the first N_k words of the decryption key.

[0105] Thus, the RAM now contains the initial round key for encryption and the initial round key for decryption. Therefore, it does not matter whether the next operation to be performed by the cryptographic engine is an encryption operation or a decryption operation--the expansion processor can commence key expansion starting from either the upper half 102 or lower half 101.

[0106] During decryption, as depicted in Fig. 5, the Encryption Round Keys are applied in reverse order.

[0107] Therefore, in operation of the present invention, during decryption it is necessary to generate W (i) from W (i+N_k) and W (i+N_k-1).

[0108] The inverse of the key expansion process requires that:

[0109] Rule 1: $W\ (i{-}N_k) = W\ (i) \oplus W\ (i{-}1)$

[0110] for all i, except:

[0111] Rule 2: $W\ (i{-}N_k) = W\ (i) \oplus \mathrm{SubWord}\ (\mathrm{RotWord}\ (W\ (i{-}1))) \oplus R_{con}\ (i/N_k)$

[0112] when i mod N_k=0, and

[0113] Rule 3: $W\ (i{-}N_k) = W\ (i) \oplus \mathrm{SubWord}\ (W\ (i{-}1))$

[0114] when i mod N_k=4 and N_k=8.

[0115] Note, that all W (i-N_k) and W (i) have interchanged places, but the complex second input is the same as for encryption.

[0116] Taking N_k=4 as an example, the last W that was generated during encryption was W (43). During decryption key expansion, the first time W is loaded, it is loaded from RAM 101; thereafter subsequent W may be obtained from W_{tmp} by means of multiplexer 125.

[0117] Thus, the first step is to load W with W (43) (found in the upper RAM half 102 at W_{11}, OffSetCnt 3) and W_{tmp} with W (42) (found in the upper RAM half 102 at W_{10}, OffSetCnt 2). Then, we calculate $W\ (38) = W\ (42) \oplus W\ (41)$ and write the result to RAM 101 in the lower half 103 at W_3. The content of W_{tmp} is then shifted to W, which then holds W (42) and W_{tmp} is loaded with W (41).

[0118] In the next cycle, we calculate $W(39) = W(43) \oplus W(42)$ and write the result to RAM 101 at W_1 and we shift the content of Wtmp to W, which then holds W(41) and we load Wtmp with W(40). This cycle is repeated for successive W.

[0119] In general, register W is loaded from RAM (or from W_{tmp}) with W(i) and register W_{tmp} is loaded from RAM with W(i-1). Then W(i-N_k) is calculated and stored in lower RAM half at position W_i mod 8 and the content of W_{tmp} transferred to W.

[0120] The decryption key expansion process runs in parallel with the decryption processor which preferably works word-by-word rather than on blocks 128 bits wide, i.e. the content of W is also passed to the decryption engine 140 for use as input for the decryption operation.

[0121] At the start, the following initialization settings apply:

[0122] OffSetCnt=3, OffSetHiRd=1, OffSetHiWr=0, RndCnt=4N_r+3.

[0123] The RAM 101 is read at address OffSetCnt [OffSetCnt=3, giving W (4N_r+3), e.g. W (43) for N_k=4] and stored in W.

[0124] Then, the following procedure is executed N_k-1 times:

[0125] 1. Read the RAM at $W_{OffSetCnt-1}$ mod N_k from the upper half and store it in Wtmp [W (42), W (41) and W (40) for N_k=4].

[0126] 2. Generate the next expanded key word and write it to RAM at OffSetCnt in the lower half [W (39), W (38) and W (37) for N_k=4].

[0127] 3. Transfer the content of W_{tmp} to W

[0128] 4. Decrement OffSetCnt and decrement RndCnt.

[0129] All words from the upper half have now been used. OffSetHiRd is set to 0, so all following key words are read from the lower half. For example, for N_k=4, the memory at address 3 in the upper half contains W (39).

[0130] Now, the following procedure is executed repeatedly until RndCnt=N_k-1.

[0131] 1. Read the RAM at $W_{OffSetCnt-1}$ mod N_k from the lower half and store it in Wtmp.

[0132] 2. Generate the next Round Key word and write it to Wtmp and to the memory at OffSetCnt in the lower half.

[0133] 3. Transfer the content of W_{tmp} to W.

[0134] 4. Update R_{con} when OffSetCnt=0

[0135] 5. Decrement both OffSetCnt and RndCnt.

[0136] At this point, the last N_k round key words are used by the decryption processor 140 but we do not need to generate more Round Key words. Thus, the following procedure is executed repeatedly until RndCnt=0:

[0137] 1. Read the memory at $W_{OffSetCnt-1}$ mod Nk from the lower half and store it in Wtmp.

[0138] 2. Transfer the content of Wtmp to W.

[0139] 3. Decrement both OffSetCnt and RndCnt.

[0140] Note that the very last read may be omitted, since it will not be used.

[0141] In a preferred embodiment, the SubWord function 55, 155 in the key expansion process may be implemented by the same hardware as that which implements the SubBytes transform 20, 31 of the encryption/decryption processes. In practice, it is found that this has minimal if any delaying effect on the encryption/decryption processes. Only every Nth round, will the key expansion processor compete with the encryption/decryption process for the same hardware.

[0142] Where the key expansion and cryptographic processes are in lock step on a word-by-word basis, the key expansion engine and the cryptographic engine will wait for each other before going to the next round, and every Nth round they have also to wait for separate access to the S-Box transform functions. However, while the cryptographic engine performs the ShiftRow transform 21 or the MixColumn transform 22, the key expansion processor can use the S-Box hardware.

[0143] The minimum amount of memory 101 required for efficient bi-directional operation is $2N_k$ words: one half (N_k) to store the encryption key and the other half to store the decryption key.

[0144] During encryption, as shown in Fig. 4, the first N_k words are taken from the encryption (lower) half. All generated round key words are written to the decryption (upper) half. At the end of encryption, the decryption (upper) half holds the decryption key.

[0145] During decryption, as shown in Fig. 5, the first N_k words are taken from the decryption (upper) half, which is in effect the "initial key" for decryption. All generated round key words are written to the encryption (lower) half. Although that means that the encryption key is temporarily overwritten, after decryption, the encryption key is regenerated. The decryption key is not overwritten.

[0146] Thus, after a first encryption process, the key expansion processor can immediately generate an expanded encryption key or an expanded decryption key, by selecting to start either from the lower half 103 or the upper half 102. For first time operation, with a new key, it is necessary to perform an encryption operation in order to generate the decryption key.

[0147] It is possible to reduce the amount of memory to as little as N_k words. However, this is less efficient in that if a number of consecutive encryption or decryption operations are required, each one must be interspersed with a dummy decryption or encryption operation to regenerate the initial encryption (or decryption) key. In general, this is less desirable.

[0148] State machine 106 controls the various registers and counters as follows, applicable to all cases of N_k=4, 6 or 8.

[0149] The 3-bit up/down counter OffSetCnt 111 points the address to each half of the memory. It counts up during encryption; when it reaches N_k-1, then it is reset to 0 again. It counts down during decryption. When it is 0, it is reset to N_k-1.

[0150] When OffSetCnt=0, then Rule 2 for W (i) applies. When OffSetCnt=4 and N_k=8, then Rule 3 applies. For all other values of OffSetCnt, Rule 1 applies.

[0151] The 1-bit variable OffSetHiRd is set to point initially (for the first N_k reads) to the lower RAM half during encryption, then to the upper RAM half 102 for all subsequent reads. During decryption, OffSetHiRd is set to point initially (for the first N_k reads) to the upper RAM half then to the lower RAM half 103 for all subsequent reads. The 1-bit variable OffSetHiWr is set to point to the upper RAM half 102 for all writes during encryption, and to point to the lower RAM half for all writes during decryption. The 6-bit down counter RndCnt 110 counts the number of rounds.

[0152] With reference again to FIG. 2, the round constant R_{con} 58 must be updated (step 59) each cycle, i.e. after each use thereof.

[0153] For the first cycle, R_{con} [1] =1. After each cycle, the value of R_{con} is updated such that:

R_{con} [i/N_k] =xtime (R_{con} [i/N_k-1],

[0154] i.e. the previous value of R_{con} is left-shifted, and when the most significant bit=1 then the hex value 1B is added to R_{con}.

[0155] According to the AES specification, the function R_{con} [i/N_k] is called when

[0156] i mod N_k=0, while $Nk \leq i < N_b$ (N_r+1).

N_k	N_b	N_r	$N_b(N_r+1)$
4	4	10	44
6	4	12	52
8	4	14	60

[0157] For N_k=4, R_{con} [i/N_k] is called for at i=4, 8, . . . 40, i.e. 10 times. The last value=36h.

[0158] For N_k=6, R_{con}[i/N_k] is called for at i=6, 12, . . . 48, i.e. 8 times. The last value=80h.

[0159] For N_k=8, R_{con}[i/N_k] is called for at i=8, 16, . . . 56, i.e. 7 times. The last value=40h.

i/N_k	1	2	3	4	5	6	7	8	9	10
Rcon[i/N_k]	01	02	04	08	10	20	40	80	1B	36

[0160] In a preferred embodiment, the R_{con} function 58, 59 is implemented as an 8-bit shift register, which can shift both left (for encryption) and right (for decryption). The shift register can be preset to the following values 01h, 1Bh, 36h, 80h and 40h.

[0161] For encryption, as shown in Fig. 4, it is preset to 01h. It shifts to the left, except when it reaches 80h, at which point it is preset to 1Bh.

[0162] For decryption, as shown in Fig. 5, it is preset to 36h for N_k=4, 80h for N_k=6 and 40h for N_k=8. It shifts to the right, except when it reaches 1Bh, at which point it is preset to 80h.

[0163] Thus, the shift register effectively has three control inputs. A first control input causes a left shift (bit rotation) of the register, which is used during each cycle during the encryption key expansion. A second control input causes a right shift (bit rotation) of the register, which is used during each cycle during the decryption key expansion. A third control input causes presetting of the register with one of a number of predetermined values, according to the current value of the register, and the direction (encryption or decryption).

[0164] It will be noted, in a general sense, that the present invention provides a method of generating successive round key words of an expanded key, from an initial key, which method maintains the generated successive round key words in memory substantially only as long as they are required for use in the generation of successive round key words and for use in the parallel operation of a cryptographic process.

[0165] In the preferred embodiment, the initial key words are also maintained in the memory.

[0166] Other embodiments are intentionally within the scope of the accompanying claims.